Contents

Accessing External Services

11 minute read ✓ page test

Because all outbound traffic from an Istio-enabled pod is redirected to its sidecar proxy by default, accessibility of URLs outside of the cluster depends on the configuration of the proxy. By default, Istio configures the Envoy proxy to pass through requests for unknown services. Although this provides a convenient way to get started with Istio, configuring stricter control is usually preferable.

This task shows you how to access external services in three different ways:

1. Allow the Envoy proxy to pass requests through to services that are not configured inside the mesh.
2. Configure service entries to provide controlled access to external services.
3. Completely bypass the Envoy proxy for a specific range of IPs.

Before you begin

- Set up Istio by following the instructions in the Installation guide. Use the `demo` configuration profile or otherwise enable Envoy's access logging.
- Deploy the sleep sample app to use as a

test source for sending requests. If you have automatic sidecar injection enabled, run the following command to deploy the sample app:

```
$ kubectl apply -f @samples/sleep/sleep.yaml@
```

Otherwise, manually inject the sidecar before deploying the `sleep` application with the following command:

```
$ kubectl apply -f <(istioctl kube-inject -f @samples/sleep/sleep.yaml@)
```

> ⓘ You can use any pod with `curl` installed as a test source.

- Set the `SOURCE_POD` environment variable to the name of your source pod:

```
$ export SOURCE_POD=$(kubectl get pod -l app=sleep -o jsonpath='{.items..metadata.name}')
```

Envoy passthrough to external services

Istio has an installation option, `meshConfig.outboundTrafficPolicy.mode`, that configures the sidecar handling of external services, that is, those services that are not defined in Istio’s internal service registry. If this option is set to `ALLOW_ANY`, the Istio proxy lets calls to unknown services pass through. If the option is set to `REGISTRY_ONLY`, then the Istio proxy blocks any host without an HTTP service or service entry defined within the mesh. `ALLOW_ANY` is the default value, allowing you to start evaluating Istio quickly, without controlling access to external services. You can then decide to configure access to external services later.

1. To see this approach in action you need to ensure that your Istio installation is configured with the `meshConfig.outboundTrafficPolicy.mode` option set to `ALLOW_ANY`. Unless you explicitly set it to `REGISTRY_ONLY` mode when you installed Istio, it is probably enabled by default.

   Run the following command to verify that `meshConfig.outboundTrafficPolicy.mode` option is set to `ALLOW_ANY` or is omitted:

   ```
   $ kubectl get istiooperator installed-state -n
   istio-system -o jsonpath='{.spec.meshConfig.out
   boundTrafficPolicy.mode}'
   ALLOW_ANY
   ```

You should either see `ALLOW_ANY` or no output (default `ALLOW_ANY`).

> ⓘ If you have explicitly configured `REGISTRY_ONLY` mode, you can change it by rerunning your original `istioctl install` command with the changed setting, for example:
>
> ```
> $ istioctl install <flags-you-used-to-install-Istio> --set meshConfig.outboundTrafficPolicy.mode=ALLOW_ANY
> ```

2. Make a couple of requests to external HTTPS services from the `SOURCE_POD` to confirm successful `200` responses:

```
$ kubectl exec "$SOURCE_POD" -c sleep -- curl -
sSI https://www.google.com | grep  "HTTP/"; kub
ectl exec "$SOURCE_POD" -c sleep -- curl -sI ht
tps://edition.cnn.com | grep "HTTP/"
HTTP/2 200
HTTP/2 200
```

Congratulations! You successfully sent egress traffic from your mesh.

This simple approach to access external services, has the drawback that you lose Istio monitoring and control for traffic to external services. The next section shows you how to monitor and control your mesh’s access to external services.

Controlled access to external services

Using Istio `ServiceEntry` configurations, you can access any publicly accessible service from within your Istio cluster. This section shows you how to configure access to an external HTTP service, httpbin.org, as well as an external HTTPS service, www.google.com without losing Istio’s traffic monitoring and control features.

Change to the blocking-by-default policy

To demonstrate the controlled way of enabling access to external services, you need to change the `meshConfig.outboundTrafficPolicy.mode` option from the `ALLOW_ANY` mode to the `REGISTRY_ONLY` mode.

> You can add controlled access to services that are already accessible in `ALLOW_ANY` mode. This way, you can start using Istio features on some external services without blocking any others. Once you’ve configured all of your services, you can then switch the mode to `REGISTRY_ONLY` to block any other unintentional accesses.

1. Change the `meshConfig.outboundTrafficPolicy.mode` option to `REGISTRY_ONLY`.

   If you used an `IstioOperator` CR to install Istio, add the following field to your configuration:

```
spec:
  meshConfig:
    outboundTrafficPolicy:
      mode: REGISTRY_ONLY
```

Otherwise, add the equivalent setting to your original `istioctl install` command, for example:

```
$ istioctl install <flags-you-used-to-install-I
stio> \
                   --set meshConfig.outboundTra
fficPolicy.mode=REGISTRY_ONLY
```

2. Make a couple of requests to external HTTPS services from `SOURCE_POD` to verify that they are now blocked:

```
$ kubectl exec "$SOURCE_POD" -c sleep -- curl -
sI https://www.google.com | grep  "HTTP/"; kube
ctl exec "$SOURCE_POD" -c sleep -- curl -sI htt
ps://edition.cnn.com | grep "HTTP/"
command terminated with exit code 35
command terminated with exit code 35
```

⚠ It may take a while for the configuration change to propagate, so you might still get successful connections. Wait for several seconds and then retry the last command.

Access an external HTTP service

1. Create a `ServiceEntry` to allow access to an external HTTP service.

   > `DNS` resolution is used in the service entry below as a security measure. Setting the

resolution to `NONE` opens a possibility for attack. A malicious client could pretend that it’s accessing `httpbin.org` by setting it in the `HOST` header, while really connecting to a different IP (that is not associated with `httpbin.org`). The Istio sidecar proxy will trust the HOST header, and incorrectly allow the traffic, even though it is being delivered to the IP address of a different host. That host can be a malicious site, or a legitimate site, prohibited by the mesh security policies.

With `DNS` resolution, the sidecar proxy will ignore the original destination IP address and direct the traffic to httpbin.org, performing a DNS query to get an IP address of httpbin.org.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: httpbin-ext
spec:
  hosts:
  - httpbin.org
  ports:
  - number: 80
    name: http
    protocol: HTTP
  resolution: DNS
  location: MESH_EXTERNAL
EOF
```

2. Make a request to the external HTTP service from SOURCE_POD:

```
$ kubectl exec "$SOURCE_POD" -c sleep -- curl -
sS http://httpbin.org/headers
{
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    ...
    "X-Envoy-Decorator-Operation": "httpbin.org
:80/*",
    ...
  }
}
```

Note the headers added by the Istio sidecar proxy: `X-Envoy-Decorator-Operation`.

3. Check the log of the sidecar proxy of `SOURCE_POD`:

```
$ kubectl logs "$SOURCE_POD" -c istio-proxy | t
ail
[2019-01-24T12:17:11.640Z] "GET /headers HTTP/1
.1" 200 - 0 599 214 214 "-" "curl/7.60.0" "17fd
e8f7-fa62-9b39-8999-302324e6def2" "httpbin.org"
 "35.173.6.94:80" outbound|80||httpbin.org - 35
.173.6.94:80 172.30.109.82:55314 -
```

Note the entry related to your HTTP request to `httpbin.org/headers`.

Access an external HTTPS service

1. Create a `ServiceEntry` to allow access to an external HTTPS service.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: google
spec:
  hosts:
  - www.google.com
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  resolution: DNS
  location: MESH_EXTERNAL
EOF
```

2. Make a request to the external HTTPS service from `SOURCE_POD`:

   ```
   $ kubectl exec "$SOURCE_POD" -c sleep -- curl -sSI https://www.google.com | grep  "HTTP/"
   HTTP/2 200
   ```

3. Check the log of the sidecar proxy of `SOURCE_POD`:

   ```
   $ kubectl logs "$SOURCE_POD" -c istio-proxy | tail
   [2019-01-24T12:48:54.977Z] "- - -" 0 - 601 17766 1289 - "-" "-" "-" "-" "172.217.161.36:443" outbound|443||www.google.com 172.30.109.82:59480 172.217.161.36:443 172.30.109.82:59478 www.google.com
   ```

   Note the entry related to your HTTPS request to `www.google.com`.

Manage traffic to external services

Similar to inter-cluster requests, Istio routing rules can also be set for external services that are accessed using `ServiceEntry` configurations. In this example, you set a timeout rule on calls to the `httpbin.org` service.

1. From inside the pod being used as the test source, make a *curl* request to the `/delay` endpoint of the httpbin.org external service:

   ```
   $ kubectl exec "$SOURCE_POD" -c sleep -- time c
   url -o /dev/null -sS -w "%{http_code}\n" http:/
   /httpbin.org/delay/5
   200
   real    0m5.024s
   user    0m0.003s
   sys     0m0.003s
   ```

   The request should return 200 (OK) in approximately 5 seconds.

2. Use `kubectl` to set a 3s timeout on calls

to the `httpbin.org` external service:

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin-ext
spec:
  hosts:
    - httpbin.org
  http:
  - timeout: 3s
    route:
      - destination:
          host: httpbin.org
        weight: 100
EOF
```

3. Wait a few seconds, then make the *curl* request again:

```
$ kubectl exec "$SOURCE_POD" -c sleep -- time c
url -o /dev/null -sS -w "%{http_code}\n" http:/
/httpbin.org/delay/5
504
real    0m3.149s
user    0m0.004s
sys     0m0.004s
```

This time a 504 (Gateway Timeout) appears after 3 seconds. Although httpbin.org was waiting 5 seconds, Istio cut off the request at 3 seconds.

Cleanup the controlled access to external services

```
$ kubectl delete serviceentry httpbin-ext google
$ kubectl delete virtualservice httpbin-ext --ignore-not-found=true
```

Direct access to external services

If you want to completely bypass Istio for a specific IP range, you can configure the Envoy sidecars to prevent them from intercepting external requests. To set up the bypass, change either the `global.proxy.includeIPRanges` or the `global.proxy.excludeIPRanges` configuration option and update the `istio-sidecar-injector` configuration map using the `kubectl apply` command. This can also be configured on a pod by setting corresponding annotations such as `traffic.sidecar.istio.io/includeOutboundIPRanges`. After updating the `istio-sidecar-injector` configuration, it affects all future application pod deployments.

> Unlike Envoy passthrough to external services, which uses the `ALLOW_ANY`

traffic policy to instruct the Istio sidecar proxy to passthrough calls to unknown services, this approach completely bypasses the sidecar, essentially disabling all of Istio’s features for the specified IPs. You cannot incrementally add service entries for specific destinations, as you can with the `ALLOW_ANY` approach. Therefore, this configuration approach is only recommended as a last resort when, for performance or other reasons, external access cannot be configured using the sidecar.

A simple way to exclude all external IPs from being redirected to the sidecar proxy is to set the `global.proxy.includeIPRanges` configuration option to the IP range or ranges used for internal cluster services. These IP range values depend on the platform where your cluster runs.

Determine the internal IP ranges for your platform

Set the value of `values.global.proxy.includeIPRanges` according to your cluster provider.

IBM Cloud Private

1. Get your `service_cluster_ip_range` from IBM Cloud Private configuration file under `cluster/config.yaml`:

```
$ grep service_cluster_ip_range cluster/config.yaml
```

The following is a sample output:

```
service_cluster_ip_range: 10.0.0.1/24
```

2. Use `--set values.global.proxy.includeIPRanges="10.0.0.1/24"`

IBM Cloud Kubernetes Service

Use `--set values.global.proxy.includeIPRanges="172.30.0.0/16\,172.21.0.0/16\,10.10.10.0/24"`

Google Container Engine (GKE)

The ranges are not fixed, so you will need to run the `gcloud container clusters describe` command to determine the ranges to use. For example:

```
$ gcloud container clusters describe XXXXXXX --zone=XXXXXX | grep -e clusterIpv4Cidr -e servicesIpv4Cidr
clusterIpv4Cidr: 10.4.0.0/14
servicesIpv4Cidr: 10.7.240.0/20
```

Use `--set values.global.proxy.includeIPRanges="10.4.0.0/14\,10.7.240.0/20"`

Azure Container Service(ACS)

Use `--set values.global.proxy.includeIPRanges="10.244.0.0/16\,10.240.0.0/16`

Minikube, Docker For Desktop, Bare Metal

The default value is `10.96.0.0/12`, but it’s not fixed. Use the following command to determine your actual value:

```
$ kubectl describe pod kube-apiserver -n kube-system | grep 'service-cluster-ip-range'
      --service-cluster-ip-range=10.96.0.0/12
```

Use `--set values.global.proxy.includeIPRanges="10.96.0.0/12"`

Configuring the proxy bypass

> ⚠ Remove the service entry and virtual service previously deployed in this guide.

Update your `istio-sidecar-injector` configuration map using the IP ranges specific to your platform. For example, if the range is 10.0.0.1/24, use the following command:

```
$ istioctl install <flags-you-used-to-install-Istio
> --set values.global.proxy.includeIPRanges="10.0.0
.1/24"
```

Use the same command that you used to install Istio and add `--set values.global.proxy.includeIPRanges="10.0.0.1/24"`.

Access the external services

Because the bypass configuration only affects new deployments, you need to terminate and then redeploy the `sleep` application as described in the Before you begin section.

After updating the `istio-sidecar-injector` configmap and redeploying the `sleep` application, the Istio sidecar will only intercept and manage internal requests within the cluster. Any external request bypasses the sidecar and goes straight to its intended destination. For example:

```
$ kubectl exec "$SOURCE_POD" -c sleep -- curl -sS h
ttp://httpbin.org/headers
{
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    ...
  }
}
```

Unlike accessing external services through HTTP or HTTPS, you don’t see any headers related to the Istio sidecar and the requests sent to external services do not appear in the log of the sidecar. Bypassing the Istio sidecars means you can no longer monitor the access to external services.

Cleanup the direct access to external services

Update the configuration to stop bypassing sidecar proxies for a range of IPs:

```
$ istioctl install <flags-you-used-to-install-Istio
>
```

Understanding what happened

In this task you looked at three ways to call external services from an Istio mesh:

1. Configuring Envoy to allow access to any external service.
2. Use a service entry to register an accessible external service inside the mesh. This is the recommended approach.

3. Configuring the Istio sidecar to exclude external IPs from its remapped IP table.

The first approach directs traffic through the Istio sidecar proxy, including calls to services that are unknown inside the mesh. When using this approach, you can’t monitor access to external services or take advantage of Istio’s traffic control features for them. To easily switch to the second approach for specific services, simply create service entries for those external services. This process allows you to initially access any external service and then later decide whether or not to control access, enable traffic monitoring, and use traffic control features as needed.

The second approach lets you use all of the same Istio service mesh features for calls to services inside or outside of the cluster. In this task, you learned how to monitor access to external services and set a timeout rule for calls to an external service.

The third approach bypasses the Istio sidecar proxy, giving your services direct access to any external server. However, configuring the proxy this way does require cluster-provider specific knowledge and configuration. Similar to the first approach, you also lose monitoring of access to external services and you can’t apply Istio features on traffic to external services.

Security note

Note that configuration examples

> ⚠ in this task **do not enable secure egress traffic control** in Istio. A malicious application can bypass the Istio sidecar proxy and access any external service without Istio control.

To implement egress traffic control in a more secure way, you must direct egress traffic through an egress gateway and review the security concerns described in the additional security considerations section.

Cleanup

Shutdown the sleep service:

```
$ kubectl delete -f @samples/sleep/sleep.yaml@
```